PDP-8 PROGRAMMING

R.W. SOUTHERN

A self-instructional workbook
for octal and assembly language programming
of the PDP-8 family of computers.

Contents

A — Binary-Octal Conversion
B — Page Zero Addressing
C — Direct Addressing
D — Indirect Addressing
E — Subroutines
F — Input/Output
G — Program Interrupt

Algonquin College Bookstore
Ottawa, Ont., Canada

Algonquinote #4

PDP-8* PROGRAMMING

OBJECTIVE OF THIS WORKBOOK Upon completion of this workbook the student will be able to program a PDP-8 computer in octal and in PAL III* assembly language. Language fundamentals rather than programming techniques are emphasized.

FOR WHOM This workbook was originally written for students in the third year of their post-high school technology program at Algonquin College. It also has been used successfully by engineers, technologists, technicians, university students, high school students and by one very capable Grade 7 student. There are no special prerequisites for completion of this workbook.

VALIDATION Revision of this workbook took place after each of six group trials, involving a total of two hundred students. One group of thirty students worked through sections A, B and C without an instructor and then attempted an assignment involving "octal add and store operations on a non-zero page". Over 90% of the resulting programs were correct.

RELATED TEXT This workbook leads into the Digital Equipment Corporation textbook "Introduction to Programming" where programming techniques are further explored.

USE OF THIS WORKBOOK This workbook is best utilized if the student works through an assigned chapter on his own and then, with an instructor's guidance, writes short programs employing the newly learned material.

AVAILABLE FROM Algonquin College Bookstore,
1385 Woodroffe Ave.,
Ottawa, Ontario, Canada. K2G1V8

Copyright © R.W. Southern 1971

*Trademark Digital Equipment Corp., Maynard, Mass. U.S.A.

To The Student:

These programmed notes are for your use at your own pace. Take your time, proceeding to the next paragraph when you are satisfied with your answer after comparison with the answer given.

To use these notes effectively: -

(a) Cover the given answer shown below the horizontal line following each question. A data card is very convenient for this.

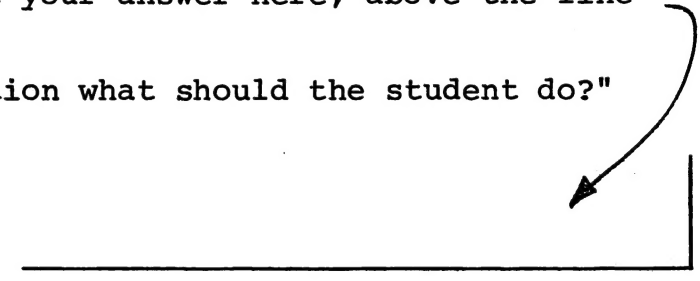
(b) Read the text material given in the frame.

(c) Write your answer to the question asked.

(d) Compare your answer with the answer given and when you fully understand any differences, if any, proceed to the next paragraph.

For practice attempt the following question, after covering the answer below the line. Write your answer here, above the line

"After answering the question what should the student do?"



Answer: The student should compare his answer with the given one in the workbook and when satisfied with any differences, move on to the next paragraph.

OCTAL AND BINARY NUMBERS

If you can convert from octal to binary and binary to octal, attempt the question below. If this is not familiar to you, go directly to frame #2 on the next page.

- Convert the following octal numbers to binary code.

  (a) 6354
  (b) 2176
  (c) 3462

- Convert the following binary numbers to octal.

  (d) 101110100010
  (e) 10011110111
  (f) 1010101100

Cover the answer shown below this line.

Answer

(a) 110011101100
(b) 010001111110
(c) 011100110010
(d) 5642
(e) 2367
(f) 1254

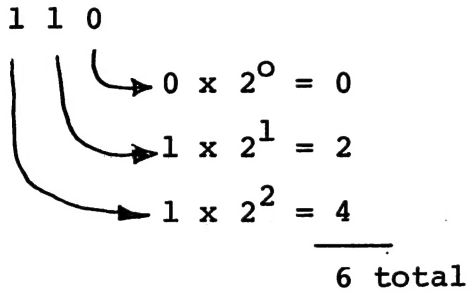
If you have any errors, proceed to frame #2. If you have no errors, skip ahead to frame B-1

The first eight binary numbers, including zero, are:-

000	=	0	100	=	4
001	=	1	101	=	5
010	=	2	110	=	6
011	=	3	111	=	7

In octal notation the largest single digit is 7, the next larger than 7 being the two digit number 10. To designate 10 as an octal number the format 10_8 is often used.

If the octal number 6 is to be converted to binary, the answer is 110, that is:-

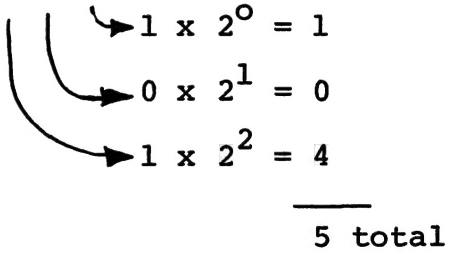


Write the binary equivalent of the octal number 5 without looking at the above.

Answer

5 = 101

To check 1 0 1



A-3

If an octal number comprises 2 or more octal digits, <u>each</u> digit may be directly equated or converted to a 3 bit binary number. For example, in converting 53_8 to binary 5 = 101 and 3 = 011.

Therefore $53_8 = \underbrace{101}_{5}\underbrace{011}_{3} = 101011$

Write the binary equivalent of 27_8

$27_8 = 010111$ = To check $\underbrace{010}_{2}\underbrace{111}_{7}$

A-4

What is the binary equivalent of 6352_8?

6352 = 110011101010 To check $\underbrace{110}_{6}\underbrace{011}_{3}\underbrace{101}_{5}\underbrace{010}_{2}$

A-5

Convert 5104_8 to binary.

$5104_8 = 101001000100$

BINARY TO OCTAL CONVERSION

To convert a binary number to octal the binary number must be broken into groups of 3 bits each, beginning from the right or least significant bit. Each 3 bit group can then be directly converted to its octal equivalent. Given 010111 the two "3 bit" blocks are easily formed, resulting in the octal number 27.

If you wish to check, reconvert this number to binary.

Convert the binary number 110101 to octal.

$110101 = 65_8$ To check $\overbrace{110}^{6}\ \overbrace{101}^{5}$

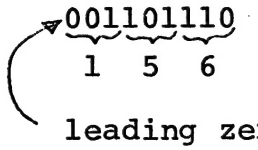
If the number of bits in a binary number is not a multiple of 3, leading zero are added to fill out the left group of 3 bits.

eg. 10101 = 010101 = what octal number?

25_8 To check $\overbrace{010}^{2}\overbrace{101}^{5}$

Convert 01101110 to octal

156_8



leading zero is added.

A-9

Convert 101110100010 to octal.

5642_8

A-10

Convert 11110100101 to octal

3645_8 Note that a leading zero was added to the binary number.

A-11

Now go to frame A-1 and try the question there.
In the answer are directions to your next frame.

PDP-8 PROGRAMMING FUNDAMENTALS

Objective: In this section you will write short programs involving basic arithmetic operations such as addition and multiplication. This will involve the use of operation codes and computer memory addresses.

The PDP-8 family of computers uses a 12 bit number to designate the instructions which the computer is to execute. The first 3 bits form the Operation Code or "Op Code" which designates WHAT the computer is to do. For most of the instructions the remaining 9 bits, the address, designate WHERE in the computer's memory that data comes from or goes to. Open up Panel B-1 (at the back of these notes) where the Op Codes are listed. Note that the op code for addition is 001, in binary format. The op code for "Deposit and clear the Accumulator" is 011 or 3_8.

How many different op codes can be designated using a 3 bit number, using 000 too?

8 (000 to 111)

B-2

A typical 12 bit binary instruction, 001000101010, is awkward to use in binary format. Express this binary number in octal format and designate the name for the first octal digit.

op code (also called the operator)

1052_8

The octal number 0052 represents the memory address from which data will be obtained, and then added to the accumulator. The accumulator (AC) itself is a 12 bit register where arithmetic operations take place.

B-3

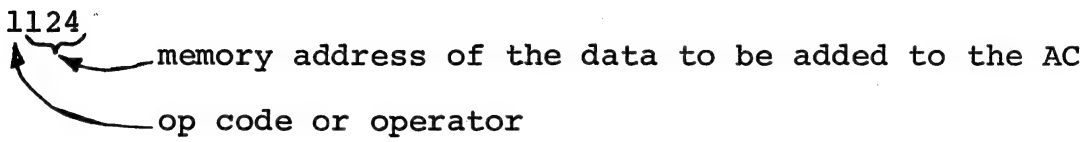
What happens when the instruction 3126 is executed?

The contents of the accumulator (AC) is deposited in memory address 126 and the AC is then cleared.

3 = deposit in memory and clear the AC
0126 = memory address where the data is deposited

B-4

Assume that the AC is cleared initially. Write an instruction to add to the AC the contents of memory address 0124.



B-5

Write two instructions which, when executed sequentially, will add the contents of memory address 0047 to the AC and then add the contents of memory address 126 to the AC.

1047 ← 1 = add, 0047 = the address from which data is obtained for the addition
1126

Add to the program below the instruction which will store the sum in memory address 0051. Refer to Panel B-1 at the back of these notes if necessary.

1047

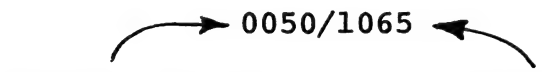
1126

1047

1126

3051 ← Deposits or stores the contents of the AC, where the addition was performed, in memory address 0051, then clears the AC.

To enable the computer to execute a program the instructions forming the program must themselves be stored in memory. The following format is used to specify the fact that memory address 0050 contains 1065, an instruction.



- Specify that memory address 0100 contains the instruction, 1047.

0100/1047

The next instruction executed by the computer after 0100/1047 will be the instruction in the next address, 0101 in this example. If the next instruction is to "add the contents of address 0126 to the AC", specify the instruction using the above format.

0101/1126

0101 is the next address after the previously executed instruction (0100/1047)

So far the program is 0100/1047

0101/1126

Specify the next instruction which will deposit the sum, now residing in the AC, in address 0051. Use the same format as in the last 2 frames.

0102/3051

contents of address 0102, a deposit instruction

memory address 0102

A jump instruction (op code 5) permits a change in location for the next executed instruction e.g. the instruction 5062 means "Go to address 0062 and execute the instruction contained there". When that instruction is executed the next instruction normally would be contained in address 0063. In the program below specify the fourth instruction which causes a jump to location 0165.

0100/1047

0101/1126

0102/3051

0103/5165

To execute this 4 line partial program the computer would be started at memory address 0100.

Execution of the first instruction 0100/1047 causes the contents of 0047 to be added to the AC. As yet the data in location 0047 has not been defined. Using the same format as in previous frames, specify that location 0047 contains 1072

0047/1072

Note that the data address 0047 is well away from the instruction addresses in the program (0100, 0101, 0102-----). In this way no confusion should arise in determining whether a memory location contains data or an instruction, even though both are represented by 4 octal digits.

It is where the program begins which is important, in determining what is interpreted as an instruction and what is data. For example, if the computer were accidentally started at 0047, the contents, 1072, would be interpreted as an "add" instruction, with the data coming from location 0072. When the computer is started at 0100, the correct address, the contents of 0047 is used as data, as intended.

Specify that 1013 is the data to be added, when 0100/1047 is executed.

0047/1013 1013 is the data, residing in memory address 0047

0100/1047
0101/1126

The second instruction of this program is to cause 1321 to be added to the AC. Add one line to the program at the left to specify this.

0047/1013

0126/1321

The program now is:-

0100/1047 } Instructions
0101/1126 }

After 0101/1126 is executed the sum in the AC is 1013 + 1321 = 2334.

0047/1013 } Data
0126/1321 }

Note that no particular order is required in choosing a data address as long as the data address is the same as that named in the related instruction.

0100/1047
0101/1126

A third operation is to take place now:-

1123, the contents of address 0146, is to be added to the AC after the two instructions in the program are executed. Add the two necessary lines to the program.

0047/1013
0126/1321

0100/1047
0101/1126
0102/1146 ◄— New instruction, stored in 0102, the next address
0047/1013
0126/1321
0146/1123 ◄— New data, in address 0146

After these three additions are completed the computer is to store the sum, now in the AC, in address 0076 and then clear the AC. Modify the program to perform this operation.

```
0100/1047
0101/1126
0102/1146

0047/1013
0126/1321
0146/1123
```

```
0100/1047
0101/1126
0102/1146
0103/3076 <-- { 3 = store in memory, then clear the AC.
                0076 = memory address where the data is stored.
0047/1013
0126/1321
0146/1123
```


Write a program which will add 2314_8, the contents of location 0134 to the AC, then add 1460_8, the contents of location 0135 to the AC, then store the sum in location 0162 then clear the AC. The program is to start at location 0100.

```
0100/1134 }
0101/1135 } <-- Instructions
0102/3162 }
0134/2314 }
0135/1460 } <-- Data
```

In the addition operation in the previous frame assume that the AC is cleared (AC = 0000) before the program begins. What is the AC contents after?

(a) 0100/1134 is executed

(b) 0101/1135 is executed

(a) 2314

(b) 3774 (2314 + 1460)


```
0140/1037
0141/1062
0142/5150
0150/3105
0037/0023 } Data
0062/1064 }
```

What is the contents of the AC after each instruction in this program is executed? What is the last instruction executed in the program shown. Assume that the AC is zero initially.

After 0140 AC = 0023
" 0141 AC = 1107
" 0142 AC = 1107
" 0150 AC = 0000

The last instruction executed is 3105

$0023_8 + 1064_8 = 1107.$

If you have problems here, convert both numbers to binary, add and then convert the sum to octal.

Fold away Panel B-1 and open Panel B-2 at the back of these notes.

All instructions using operation code "7" are called Operate Microinstructions and are used to modify the AC contents or in some cases to permit branching in the execution of a program. The program below used some of the Group 1 Microinstructions from Panel B-2. State the contents of the AC after each instruction in the program is executed.

```
0120/7200
0121/1062
0122/7001
0123/3065
0062/1536
```

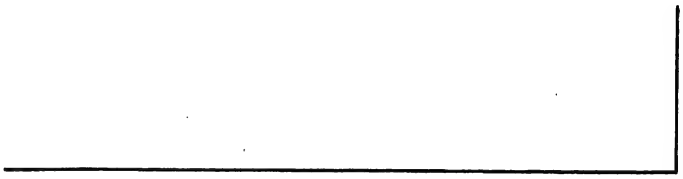
0120/7200 AC = 0 since the instruction clears the AC.
0121/1062 AC = 1536, the contents of location 0062.
0122/7001 AC = 1536 + 1 = 1537 The instruction increments the AC.
0123/3065 AC = 0000 after the AC contents is stored in 0065.

Note that no address is described in the Operate Microinstructions. All operations involve the AC.

Starting at memory address 0120 write a program which will:-

(a) Clear the AC.
(b) Load the contents of the external switch register into the AC.
(c) Complement and increment the AC (one instruction is sufficient) producing the negative or two's complement of the number in the AC.
(d) Store the number, now in the AC in its negative form, in location 0136 and then clear the AC.
(e) Halt the program.

Note: Use any Operate Microinstruction from Panel B-2.



(a) 0120/7200 (or 7600) your choice
(b) 0121/7604
(c) 0122/7041 → See "Combined Microinstructions".
(d) 0123/3136 ← For 7041 note that the complement operation inverts all AC bits (0 to 1 and 1 to 0) then the increment portion of this instruction adds 0001 to the AC.
(e) 0124/7402 ← In Group 2 Operate Microinstructions

It is standard practice to clear the AC and perhaps also the link, the "13th bit" associated with the AC, before a program starts. This will ensure that no previous data in the AC will intrude on the users program.
The halt instruction at the end is self-explanatory.
The "complement and increment" instruction permits the negative value of a number in two's complement form to be produced in the AC.

0100/7200
0101/1064
0102/3065
0103/2065
0104/5103
0105/7402
0064/7776

In this program what is the octal contents of location 0065 just after 0102/3065 has been executed during the running of the program?

0065/7776, 7776 was transferred to the AC initially and then to location 0065.

After 0103/2065 has been executed and before 0104/5103 is executed, what is the contents of 0065?

7777

Will the computer now skip the next instruction (0104/5103)?

No, since the contents of 0065 is not zero. The instruction 2065 increments the contents of 0065 to 7777. If the incremented contents had been 0000, the next instruction,0104/5103,would be skipped or omitted. Since the incremented contents (7777) is not zero the next instruction (0104/5103) is executed.

What are the next two instructions executed after 0103/2065 is executed?

0104/5103 - the instruction causes the computer to jump back to memory location 0103 and execute the instruction stored there (2065).

0103/2065 - executed now for the second time.

After the execution of 0103/2065 for the second time, what is the contents of 0065 and what instruction is executed next?
Hint: A memory address, like the AC, has 12 bits only.

0065 now contains 0000 causing the computer to skip the next instruction in 0104, and to execute the instruction contained in 0105. In this program we have a looping operation, the exit occurring only when the memory contents has been incremented from 7777 to 0000. This procedure is often used for repetitive operations, with a particular memory location acting as a "counter" to determine when the exit from the loop will occur.

0140/7200
0141/1051
0142/3052
0143/2052
0144/5143
0145/7402
0051/7774

How many times will the instruction contained in location 0143 be executed in this program?

Four times

After once 0052/7775	After the second time 0052/7776	After the third time 0052/7777	After the fourth time 0052/0000 causing a skip to the halt in 0145

```
0130/7200
0131/1045
0132/7041
0133/3046
0134/1065
0135/2046
0136/5134
0137/7402
0045/0003
0065/0002
```

What is the contents of address 0046 after 0133/3046 is executed?

7775, the two's complement of 0003 expressed in octal form.

$$\begin{array}{r} 7777 \\ -3 \\ \hline 7774 \\ +1 \\ \hline 7775 \end{array}$$

If this complementing of 0003 is not understood, refer to any text on two's complement arithmetic.

After 0135/2046 is executed once, what is the octal contents of the AC?

_____

of address 0046?

_____

AC = 0002
0046/7776

Note that the instruction 2046 increments the contents of address 0046. The AC is not changed by this instruction.

After 0135/2046 is executed once, what instruction is executed next?

_____

0136/5134, since address 0046 does not contain zero.

What is the contents of the AC when the program halts?

AC = 0006 { AC = 0002 & 0046/7776 } then { AC = 0004 & 0046/7777 } then { AC = 0006 & 0046/0000 } then halt

The program discussed in the past few frames can be modified to add any number to the AC N times, or in other words, multiply a number by N. The product cannot exceed 7777_8. Modify the program to multiply 4 by 6. Show only the changes.

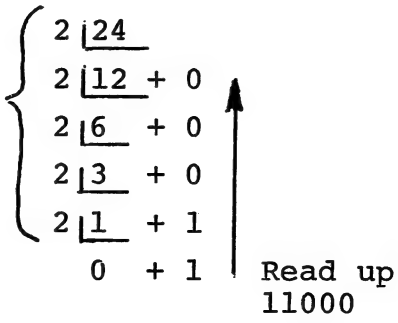
Two solutions exist (a) 4 is added 6 times
(b) 6 is added 4 times

	Changes		Changes
For (a)	0045/0006	For (b)	0045/0004
	0065/0004		0065/0006

What is the resulting product in:

(a) decimal
(b) binary
(c) octal

(a) 24
(b) 11000 ⟶
(c) 11000 = 30_8



ASSEMBLY LANGUAGE PROGRAMMING

All instructions so far have been written in binary or octal format. In longer programs, where debugging can become complicated, programs are often written in assembly language and then converted to binary format by the computer itself before execution. The assembly language used on PDP-8 computers is called PAL III (Programming Assembly Language - version 3).

For example the instruction 1047 may be expressed as TAD 0047 where TAD is the assembly language instruction for addition, corresponding to op code 1 (See Panel B-1), and 0047 is the referenced address.

Write the instruction which will add the contents of address 0137 to the AC.

TAD 0137

Write a 2 line program which will add the contents of address 0036 to the AC, then store the AC contents in address 0037. For now ignore the starting address of the program.

TAD 0036
DCA 0037

Using assembly language instructions write a 4-line program in which the AC is the first cleared, then the contents of address 0104 is made the same as the contents of address 0130, after which the program halts. This copying operation is often referred to as initializing. Refer to Panels B-1 and B-2 for assembly language instructions.

CLA
TAD 0130
DCA 0104
HLT - Note it is HLT, not HALT.

The instruction 1047 also may be expressed in the PAL III assembly language as TAD NUM

where TAD = 1 = op code for addition as noted previously.
NUM = symbolic address of the data to be added and in this example, is address 0047.

The symbolic address, NUM, in this example could just as easily have been DATA or K5 or CONSTN or any alphanumeric word as long as the first character of the word is an alphabetic character (6 characters maximum).

The instruction 5236 in PAL III format could be

JMP NEXT
or any alphanumeric word of your choice.
(6 characters maximum, first character alphabetic).

Write an instruction to store the AC contents in a symbolic address, CONTNT

DCA CONTNT

To designate the starting address of an assembly language program the program is preceded by *XXXX if the program is to begin at memory address XXXX.

For example *0200 designates that the program is to start at memory address 0200.

Write the first two lines of a PAL III program, starting at address 2200, which adds the contents of symbolic address NUM3 to the AC.

```
*2200
 TAD NUM3
```

More instructions, if added here, would be assigned to address 2201, 2202 etc.

To designate that the data contained in symbolic address NUM3 is 2326, the following format is used:

NUM3, 2326

The program now appears as:-

```
          *2200
           TAD NUM3
             |
             |
NUM3,      2326
```

Add two more lines to the program to cause 0526, the contents of symbolic address NUM4 to be added to the AC.

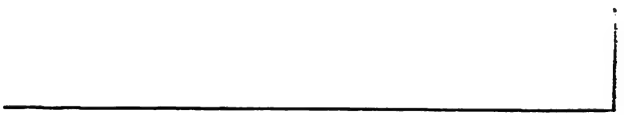
```
*2200
 TAD  NUM3
 TAD  NUM4   - the assembler automatically assigns the address 2201 to
                                                this instruction.
   |
   |
NUM3,   2326   }  Note that the data appears at the end of the instructions.
NUM4,   0526   }
```

Another line may be added to store the sum, residing in the AC, in a symbolic address called POCKET. The program with a HALT included now is: -

```
        *2200
        TAD NUM3
        TAD NUM4
        DCA POCKET
        HLT
NUM3,   2326
NUM4,   0526
POCKET, 0000
```

The symbolic address, POCKET, needs no particular initial contents, since its final value will be that obtained from the AC in this program. However, to enable the PAL III assembler to recognize POCKET as a symbolic address an arbitrary contents, usually 0000, is assigned as is seen in the last line of the program above.

Write a program, starting at address 0140, which will clear the AC, then subract "DATA1" from "VALUE" and store the result in the symbolic address "RESULT". DATA 1 = 5. VALUE = 7. If necessary, refer to frame B-27 for a procedure required in subtraction.



```
        *0140
        CLA
        TAD DATA1
        CIA
        TAD VALUE
        DCA RESULT
        HLT
DATA1,  0005
VALUE,  0007
RESULT, 0000
```

Convert the following binary program to PAL III assembly language. Choose you own symbolic addresses.

```
0050/7200
0051/1057
0052/7041
0053/3061
0054/1060
0055/3062
0056/7402
0057/2025
0060/1402
```

```
0050/7200          CLA
0051/1057          TAD    DATA1
0052/7041          CIA
0053/3061          DCA    NUM1
0054/1060          TAD    DATA2
0055/3062          DCA    NUM2
0056/7402          HLT
0057/2025    DATA1, 2025
0060/1402    DATA2, 1402
              NUM1, 0000
              NUM2, 0000
```

NOTE: Program lines linked by arrows above must have the same symbolic addresses, which are your choice.

Every program line (either instruction or data) has associated with it an octal address. Although not necessary, any instruction in a program may have a symbolic address, also referred to as a tag, as shown below. Here "AGAIN" is the tag to which the program jumps in the loop formed by { AGAIN, ISZ LOOP / JMP AGAIN

```
0114/1121            TAD NUM7
0115/3122            DCA LOOP
0116/2122     AGAIN, ISZ LOOP
0117/5116            JMP AGAIN/or any other address of
0120/7402            HLT                          your choice
0121/7774      NUM7, 7774
0122/0000      LOOP, 0000
```

It is also possible to write the instruction JMP AGAIN as JMP. -1 which is interpreted as jump back one line from the address of the present instruction.

Rewrite the incomplete binary program shown below in PAL III assembly language. Show the optional form for the "5125" instruction.

```
0123/1131
0124/3132
0125/2132
0126/2133
0127/5125
0130/7402
0131/1326
0132/0000
```

```
                     *0123
0123/1131            TAD WORD
0124/3132            DCA STORE1
0125/2132      MORE, ISZ STORE1
0126/2133            ISZ STORE2
0127/5125            JMP MORE   (or JMP.-2)
0130/7402            HLT
0131/1326      WORD, 1326
0132/0000    STORE1, 0000
0133/0000    STORE2, 0000
```

} or your choice of symbolic addresses

Convert the binary program to PAL III assembly language format. Choose your own symbolic address.

```
0100/7200
0101/1106
0102/3107
0103/2107
0104/5103
0105/7402
0106/7776
```

```
          *0100
0100/7200 CLA
0101/1106 TAD NUM
0102/3107 DCA LOOP
0103/2107 AGN, ISZ LOOP
0104/5103 JMP AGN
0105/7402 HLT
0106/7776 NUM, 7776
         LOOP, 0000
```

Convert the following program to its octal format. As a start assign an octal address to each line of the program.

```
        *0100
         CLA
         TAD NUM
         CIA
         DCA COUNT
  AGN,   TAD DATA
         ISZ COUNT
         JMP AGN
         HLT
  NUM,   0012
 DATA,   0164
COUNT,   0000
```

```
                    *0100
0100/7200           CLA
0101/1110           TAD NUM
0102/7041           CIA
0103/3112           DCA COUNT
0104/1111     AGN,  TAD DATA
0105/2112           ISZ COUNT
0106/5104           JMP AGN
0107/7402           HLT
0110/0012      NUM, 0012
0111/0164     DATA, 0164
0112/0000    COUNT, 0000
```

OBJECTIVES

In this section you will write programs similar to those in part B, but in memory addresses greater than 0177.

PRESENT PAGE ADDRESSING

Since 3 of the 12 bits of an instruction are used for the op code only 9 bits remain for the address, permitting 2^9 or 512_{10} memory addresses to be referenced. Almost all computers have at least 2^{12} memory addresses. This presents a problem, requiring a unique approach to addressing.

For this reason the memory in the PDP-8 computer is arbitrarily divided into pages, each containing 200_8 addresses eg.

Page #0 = 0000 to 0177_8

Page #1 = 0200_8 to 0377_8 (Note: $200_8 = 128_{10}$)

Page #2 = 0400_8 to 0577_8

Open up Panel C at the back of the notes where the memory pages are depicted.

If there are 2^{12} memory addresses, what is the last address on the last page?

7777_8 the largest 12 bit number, expressed in octal.

(111 111 111 111)

What is the first address on the last page?

7600 (7600 to 7777 = 200_8 locations)

C - 3

What is the first address on the page containing memory address 0637?

0600_8 since pages begin every 200_8 locations.

C - 4

What is the first address of the page on which memory address 2362 resides?

2200 The boundaries of this page are 2200 and 2377.

C - 5

A memory address can be broken into 2 parts, the sum of which equals the memory address. Using memory address 2362 as an example these parts are:-

- the starting address on the page (2200 in this example)
- the page address (2362 - 2200 = 162)

The sum of 2200 + 162 = 2362, the memory address.

Note that the page address cannot exceed 177.

What is the starting address and page address for memory address 3253?

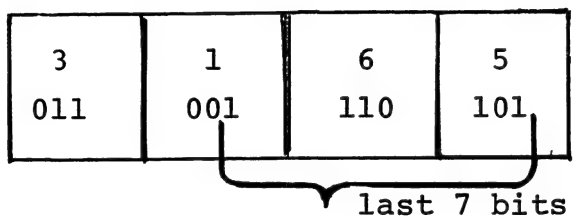
starting address = 3200
page address = 053
} sum = 3253, the memory address

What is the starting address and the page address for memory location 4725?

4600 = starting address (every 200_8 locations)
125 = page address
4725 = memory location or memory address.

C - 7

The page address, discussed in previous frames, is referred to in an instruction such as 3165_8 where the last 7 bits of the instruction form the page address, 165.



For the moment ignore what page the page address is on. In the instruction 2043, what is the page address?

043 from the last 7 bits.

C - 8

If the instruction is 5142, what is the page address?

142, again from the last 7 bits.

The page referenced in an instruction such as 3165 is determined by bit #4 of the instruction. If Bit #4 = 0, page 0 is referenced, where the memory address = 0000 + page address

(starting address of page 0)

In the instruction 5173 what is the page address and memory address referenced? Check bit #4 first. NOTE THAT BITS ARE NUMBERED FROM THE LEFT, THE FIRST BIT BEING BIT #0.

Bit #4 = 0. Therefore page 0 is referenced
Page address = 173 (Use 3 digits for the page address).
Memory address = 0173 (0000 + 173). Use 4 digits for memory addresses.

To check bit #4

5 1 7 3
101 001 111 011

Bit #4 = 0

Again note that the left bit in a 12 bit word is Bit #0

In the instruction 2145 what is the referenced memory address?

0145
001

Bit #4 = 0. Therefore page 0 is referenced.
The page address is 145. The memory address is 0145.

The above answer 0145 could have been produced many frames back without knowing about page addresses and pages, but this paging detail is necessary in the next few frames.

If now bit #4 = 1, the page address is still the last 7 bits but the page referenced is the present page, that is, the page on which the instruction itself resides. For example in 0653/3274 the instruction address 0653 is on the page which starts at 0600. The page address referenced in the instruction is 074. The memory address actually referenced is 0600 + 074 = 0674 where the data will be stored. What is the memory address referenced in the instruction 1473/1265? Check Bit #4 first.

1465 Bit #4 = 1 1473/1265

010 ← Bit #4 = 1
Therefore the present page is referenced, the page beginning at 1400.

The page address referenced in the instruction is 065 from the last seven bits.

The referenced memory address is then 1400 + 065 = 1465, where the data for the addition is contained.

If, in error, you said that bit #4 = 0, note that the first bit is bit #0. Bit #4 then is 1 in the instruction 1265.

What is the memory address referenced in the instruction 2371/1264?

2264 Bit #4 = 1. Therefore the present page, starting at 2200 is referenced.

2200 + 064 = 2264

064 → page address, from the last 7 bits.

2200 → starting address of the page on which the instruction resides.

C-13

Where in memory is the next instruction to be executed after 5156/5372 is executed? If necessary check on the meaning of op code 5.

5172 = 5000 + 172 ← page address, from the last 7 bits

↑ starting address of present page

Bit #4 = 1. Therefore the present page is addressed.

C-14

When the instruction 0657/5247 is executed, where in memory is the next instruction?

0647

The referenced address is 0647 (0600 + 047).

C-15

In the instruction 1235/3372 where is the data stored?

1372. Bit #4 = 1 1200 + 172 = 1372.

In the instruction 2356/1327 where is the data which is to be added to the AC?

2327 Bit 4 = 1 { The referenced memory address is 2200 + 127 =2327

What happens when 1756/3324 is executed?

The AC data is stored in 1724 and the AC is then cleared.

Write the instruction, stored in address 3362, which causes an addition operation, the data coming from memory address 3371.

3362/1371

The present page is referenced, therefore bit #4 = 1
The page address is 171, (3371 - 3200)

3362/1371 ← Bit #4 = 1
3 = 011 ← first Bit of 7 bit page address

Write the instruction, residing in 4126 which stores the AC data in 4037 and then clears the AC.

4126/3237 ← Bit #4 = 1 for present page
2 = 010

Write a short program beginning at address 3440 which will

- clear the AC
- then add 1326, the contents of address 3470 to the AC
- then store this data, residing in the AC, in address 3455

3440/7200
3441/1270 add the contents of page address 070 of the present page (the page which starts at 3400) to the AC
3442/3255 store the contents in address 3455 (page address 055 on the present page)
3470/1326 data

This operation is essentially a "move" operation, the data from address 3470 now appearing in address 3455. Note that the original contents of 3470 is not changed, hence the data in both addressès is identical.

Write a program starting at address 2550 which will move 6537, the contents of 2416 to address 2564.

2550/7200 the "clear" instruction is optional here.
2551/1216
2552/3364
2416/6537

Write a program starting at 4150 which will

- clear the AC
- then "move" 2156, the contents of address 0063 to address 4062
- then move it from address 4062 to address 0165
- then halt

```
4150/7200
4151/1063 -   Here page 0 is referenced, hence bit #4 = 0
4152/3262 }   Here the present page is referenced, hence
4153/1262 }   bit #4 = 1
4154/3165
4155/7402
0063/2156 -   data
```


Write a program starting at location 3640 which will

- clear the AC
- then add 1426, the contents of address 3670 to the AC
- then add 2150, the contents of address 0135 to the AC
- then store the result in address 3765
- then jump to location 0140

```
3640/7200
3641/1270
3642/1135
3643/3365
3644/5140
3670/1426
0135/2150
```

Assembly language programs may be relocated from page 0 to some non-zero page by merely changing the starting address. Modify the program below to cause the first instruction to reside in address 2650.

```
*0050
CLA
TAD START
DCA LOOP
    ¦
    ¦
```

```
*2650
CLA
TAD START
DCA LOOP
    ¦
    ¦
```


What is the AC contents when the program below is executed?

```
*1410
CLA
TAD NUM
TAD DATA5
HLT

  NUM, 0034
DATA5, 1621
```

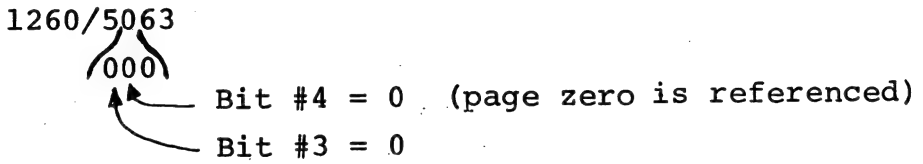
AC = 1655

0034 the contents of NUM is added to the AC; then 1621, the contents of DATA5, is further added, resulting in the sum of 1655.

INDIRECT ADDRESSING

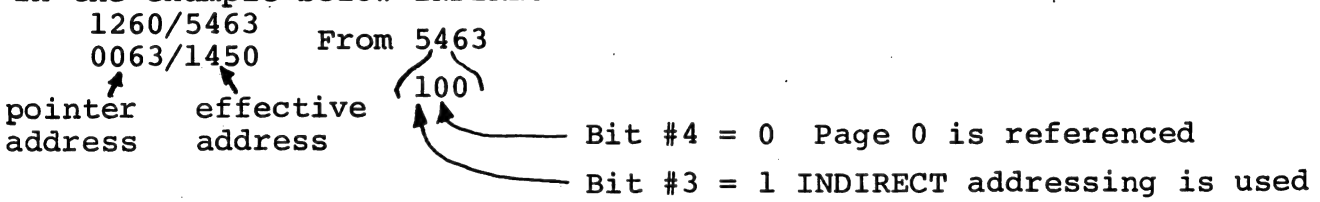
Objectives: In this section you will write programs in which the instructions reside on one page and the data on a different, non-zero page. Also you will write programs utilizing a jump to an instruction on another non-zero page. Be patient in this chapter and re-read if necessary.

So far, by using bit #4, it is possible to have the computer remain on the present page or jump to page 0. It is not possible to jump from one non-zero page to another non-zero page using the programming techniques studied so far. If you doubt this statement, try encoding the instruction residing in 1260 which will cause a jump to 1450. This latter problem is solved via indirect addressing, using bit #3. For review let us look at the instruction



This illustrates a DIRECT address change, the next instruction coming from memory address 0063.

In the example below INDIRECT ADDRESSING is used.



When bit #3 = 1 indirect addressing is required. The address 0063 obtained from the instruction 5463 is not the address to which a jump is made. 0063 is a POINTER address, which on a separate line names as its contents an EFFECTIVE address to which the jump is made. An indirect addressing operation always requires an extra line in which the pointer address contains, as its contents, the effective address.

1260/5463
0063/2627

In this example 0063 again is the pointer address. This time the true address or effective address to which the computer jumps is 2627. To complete the necessary detail in the program a line 2627/1243 would show that the instruction in 2627 causes an add operation to take place. Note that effectively the program changed from the page starting at 1200 to the page starting at 2600.

How do you know that indirect addressing is employed in the instruction 1260/5463?

Bit #3 = 1. Therefore indirect addressing is used.

1260/5463
0063/1625 ←

When 1260/5463 is executed the pointer address obtained from this instruction is 0063. What does the next line of the program tell you?

The effective address is 1625, to which the computer will indirectly jump for its next instruction.

When indirect addressing is used, the extra line relates the pointer and effective addresses.

It is also worth noting that when indirect addressing is used the second digit of the octal instruction is greater than 3, since bit #3 = 1 with indirect addressing.

1320/5572

In this indirect instruction, what is the pointer address?

0172

Bit #3 = 1. Therefore indirect addressing is used and the pointer address is then 0172.

Indirect addressing may be used for other instructions.

2246/3521
0121/2652

For this instruction, where is the data stored?

2652

From the instruction 3521 the pointer address 0121 is obtained. Memory address 0121 has as its contents the effective address 2652, the address in which the data will be stored.

In any indirect operation an extra line always must be designated, the address being the pointer address, obtained from the indirect instruction, and the contents being the effective address, where the operation will take place.

1260/5463
0063/1625
1625/7200

In this partial program, what is the next instruction executed after the one in 1260 is executed?

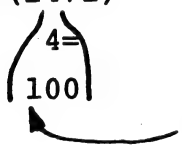
7200 = clear the AC.

Open up Panel D at the back of these notes.

2621/7200
2622/1471
2623/7402
0071/3643
3643/2653

In this miniporgram, also at the top of Panel D, is indirect addressing used in the second instruction? How do you know?

Yes (1471)



Bit #3 = 1. Therefore indirect addressing is used.

What is the pointer address?

0071,

What is the effective address?

3643, the contents of the pointer address 0071

D - 9

What happens when this instruction, 1471, is completely executed

It causes the addition of 2653, the contents of location 3643 to the AC.

D - 10

In the program shown at the bottom of Panel D, the first instruction clears the AC. What is the contents of the AC after the second instruction is completely executed?

AC = 1205 1567 means indirect addition with the pointer address, 0167, providing the effective address, 6653, from which the data is obtained for addition to the AC.

What is the result of the next instruction executed?

The contents of the AC, 1205, is stored in 2433. The instruction is interpreted as " From pointer address, 0053, obtain the effective address, 2433. Then store the data in the effective address and clear the AC."

D - 12

Modify the lower program in Panel D such that the pointer address for the addition is 0163 while the data to be added is contained in 5622.

1460/7200
1461/1563 ← Contents changed
1462/3453
1463/7402
0163/5622 ← Both address and contents changed
0053/2433
5622/1205 ← Address changed

D - 13

Modify the program <u>further</u> such that the "deposit" instruction is an indirect instruction, causing the data to be stored in location 2346 via pointer address 0135.

1462/3453 becomes 1462/3535
Delete 0053/2433 and add 0135/2346

1425/1650 What two forms of addressing does this instruction specify. Note the "6" in the instruction.

Bit #4 = 1 Therefore the present page is referenced.

Bit #3 = 1 Therefore indirect addressing is used.

Combining these two facts the pointer address in an indirect addressing instruction is on the present page. The pointer address then is 1400 + 0050 = 1450. As yet the effective address for the data to be added is not specified.

1425/1650 Add two more lines to the line at the left to specify an effective address of 2722 and data of 4326.

1425/1650	Instruction
1450/2722	1450 is the pointer address
2722/4326	2722 is the effective address containing data, 4326

The pointer address, as seen from the examples shown, can only be on page 0 or on the present page. The effective address, however, can be on any page.

Fill in the blank address below to complete a program in which

- the AC is cleared
- octal data, 2604, is then indirectly added to the AC
- the program halts

2604/7200
2605/1677
2606/7402
/4138
4138/2604

2677/4138 The instruction 1677 is interpreted as:-

"From location 077 on the present page, that is from location 2677, obtain the effective address for the data, then add the data contained in the effective address to the AC." Note that 2677 is the pointer address, pointing to the effective address, 4138, where the data, 2604, is contained.

D - 17

1250/5726 What is the pointer address associated with this indirect jump?

1326 - Both Bits #3 and #4 are 1. Therefore the pointer is on the present page.

D - 18

1250/5726 What is the result when 1250/5726 is executed?
1326/2650

The next instruction executed will be contained in location 2650.

Take a break now. You have completed the hardest part of this workbook.

Indirect addressing in PAL III is implemented via the pseudo operator "I", as shown below, where I implies indirect addressing.

```
      CLA
      TAD I GO  <-- GO is the pointer address.
GO,   2750  <-- the effective address in which resides the data to
                be added to the AC.
```

Execution results in the addition of the <u>contents</u> of address 2750 to the AC.

Using NEXT as the pointer address write the necessary portion of a program to cause a jump to address 3360, which is on a different page from the jump instruction.

```
      JMP I NEXT
NEXT, 3360
```


Write the portion of a PAL III program, residing on the same page as address 0650, which adds the contents of address 1260 to the AC. Choose your own symbolic pointer.

```
      CLA
      TAD I DATA  <-- pointer address of your choice
DATA, 1260
```

If the symbolic address NUM corresponds to address 1260, the previous frame could have been written as

```
      CLA
      TAD I DATA
DATA, NUM
```

Assuming that a symbolic address PLACE corresponds to address 2241, write the portion of a program which stores the contents in address 2241, if the program itself resides on a different page from address 2241.

What results when the program below is executed?

```
      CLA
      TAD I WORD
WORD, NUM
 NUM, 1426
```

AC = 1426

WORD is the pointer address, naming NUM as the effective address in which resides 1426, which is added to the AC. NUM can be an address on any page in memory.

What results when the program below is executed?

```
        CLA
        TAD I NUM
        DCA I PLACE

   NUM, DATA
 PLACE, 1624
  DATA, 4247
```

1624/4247

4247 is indirectly added to the AC, then stored indirectly in address 1624.

SUBROUTINES

Close Panel D and open Panel E.

A subroutine is a set of sequential instruciions to which the main program jumps via the Memory Reference Instruction 4 xxx xxx xxx. (Details in Panel E). The main difference between this instruction, "Jump to Subroutine" and the normal "jump" instruction is that after the subroutine is executed control returns to the main program, to the instruction immediately following the JMS (4 xxx xxx xxx).

In panel E the instruction 4060 calls up the subroutine, located at 0060 to 0062, which itself performs the two's complement operation on the AC contents, then returns control to the next instruction in the main program. The first address of the subroutine does not contain an instruction but is reserved for use by the computer. During the execution of the subroutine the return address in the main program is automatically stored here. Initially this first address is permitted to contain anything but usually is assigned a contents of 0000 as in Panel E.

What instruction will be executed after the subroutine is completely executed?

1403/1272, the next instruction in the main program after 4060.

The subroutine itself may or may not begin with 7200, (clear the AC) in the second line, depending on whether data, resident in the AC from the main program, is used in the subroutine, In this subroutine on Panel E, AC data from the main program is used in the two's complement subroutine. The last instruction in the subroutine is always an indirect jump to the main program via the pointer stored in the first location of the subroutine.

What is the contents of location 0060 while the subroutine is being executed?

0060/1403 1403 is the address of the next instruction to be executed when control returns to main program.

E-3

What is the result of the execution of 0062/5460?

An indirect jump is performed. The pointer address is 0060, the first address of the subroutine. The effective address is 1403. When the jump is completed 1403/1272 is then executed.

E-4

Relocate the subroutine using locations 0100, 0101 and 0102. Show the changes required to the program in Panel E.

1402/4100 ←— jump to subroutine located at 0100
0100/0000
0101/7041
0102/5500 ←— Indirect jump to main program via the pointer at 0100

E-5

While the subroutine is being executed what will be the contents of address 0100 in the modified program?

1403, the return address in the main program.

Relocate the subrcutine to memory addresses 2255 to 2257 inclusive, showing only the modifications to the program in Panel E. Do not use page zero for this modification.

Since the main program occupies 7 locations there are 121_{10} vacant addresses on the present page which could be used as pointer addresses in the indirect addressing required to get to the subroutine.

One solution is shown below.

1402/4650 ← jump indirectly to the subroutine via the pointer address 1450

1450/2255 ← pointer address/effective address (2255)

2255/0000 ← 0000 will automatically become 1403 when the subroutine is executed.

2256/7041

2257/5655 ← Indirect jump via the pointer address 2255 to the main program.

E-7

Write a subroutine starting at location 2320, which will add 1206_8 to the number already in the AC and then perform the two's complement operation on the sum.

2320/0000

2321/1310 ← likely will be different from your answer depending on the data address.

2322/7041

2323/5720 ← Jump indirectly via the pointer address 2320 to the main program.

2310/1206 ← data

E-8

Write the instruction residing at address 2253 in the main program which calls up the above subroutine. To what address in the main program does control return when the subroutine is completed?

2253/4320 The return is to address 2254 in the main program.

INPUT/OUTPUT TRANSFER

Objectives In this section you will write instructions permitting data transfer between the computer and peripheral instruments, such as the teletype.

To permit the computer to communicate with external instruments or equipment, such as the teletype, input/output transfer instructions, often called IOT instructions are used.

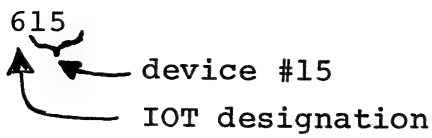
IOT instructions use op code 6, defined by bits 0 to 2. The next 6 bits (bits #3 to #8 inclusive) form the device code which determines the unique external device or instrument being addressed. For example the device code for the teleprinter is 000100 or 04_8. How many external devices can be uniquely addressed using all possible device codes, excluding 000 000 which is reserved for the program interrupt.

63_{10} or 77_8, your choice.

In many installations specific device codes are reserved for specific external devices. eg. #60, 61 and 62 are reserved for the DF32 Disk File and the software assumes these device codes.

So far the first 9 bits of an IOT instruction have been defined. An instruction 637X where X is as yet undefined says that device #37 is being addressed.

Write the first 3 octal digits of the IOT instruction which addresses device #15.



The true instruction will be in the form of 615X where X is as yet undefined.

Now open up Panel F at the back of these notes, for use in the following frames.

Associated with an IOT instruction are 3 event times in which input/output operations can take place. These event times , separated by 1.0μsec, are called event times #1 (001), #2 (010) and #4 (100) respectively and occur in that order when an IOT instruction is executed. These 3 different event times permit three different programmed operations for each external device, the event time being named by the last octal digit in the IOT instruction. For example, via the instruction 6152, a digital voltmeter responding to device code #15 could, at event time #2, read the analog voltage at its input. The same digital voltmeter via the instruction 6154 could, at event time #4, transfer digital data to the computer. What instruction would be used for the same voltmeter to cause some specific action at event time #1?

Bit #11 →

6151 = 110 001 101 001

6151
- event time #1
- device #15

event time #1 when the last 3 bits = 001 or in other words, if bit #11 = 1.

Write the octal and binary forms of the IOT instruction to cause device #35 to perform some specific operation, called operation A, at event time #1.

6351 110 011 101 001

- event time #1 (When the last digit = 1, that is if bit #11 = 1)
- device code = 35

Write the octal and binary instruction for device #35 to have it perform a different operation, operation B, at event time #4.

6354

110 011 101 100 ← for event time #4 bit #9 = 1

If it is permitted and desirable to have operation B take place 2.0μsec after operation A in device 35, then both instructions can be combined in a single IOT instruction or IOT microinstruction as it is often called. The instruction is 6355, which in binary is 110 011 101 101

Note:-

bit #11 = 1, therefore operation A takes place at event time #1

bit # 9 = 1, therefore operation B takes place at event time #4

Write an instruction to cause operation X to occur at event time #2 in device #33.

6332 — event time 2 = 010

device code

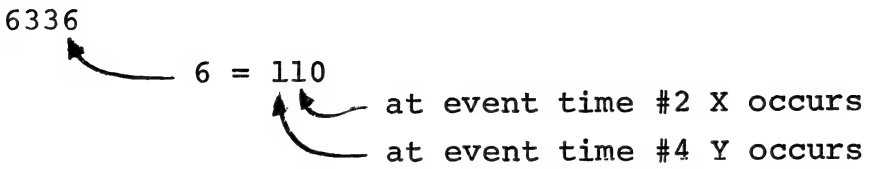
Write the instruction to cause operation Y to occur at event time #4 in the same device, #33

6334

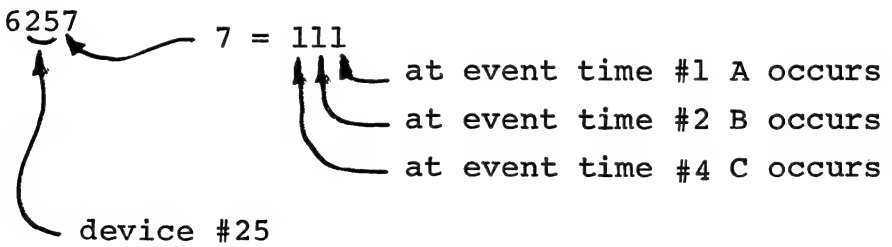
event time = 4 = 100

device code

Using the same device code as in the previous frame write the IOT microinstruction which will cause operation X to occur at event time #2 and to cause operation Y to occur 1.0μsec later at event time #4.



Write the microinstruction for device #25 which will cause operation A to occur first, then 1.0μsec later operation B, then 1.0μsec later operation C.



Specific IOT instructions produce the required command signals for the peripheral devices via input/output pulses (IOP's) which are generated in the computer and are available to the peripheral devices. If an IOT instruction calls for event time #1, eg. 6571, then a pulse, IOP1, will be sent to <u>all</u> peripheral devices, at event time #1, but will be accepted only by the addressed peripheral device, #57 in this example. IOP2 is produced if event time #2 is called for by the instruction. In this way a specific device can be commanded to perform a specific operation at a specific time.

What IOP is produced by the computer and accepted by device #35 at event time #4 by a certain instruction. Write the instruction.

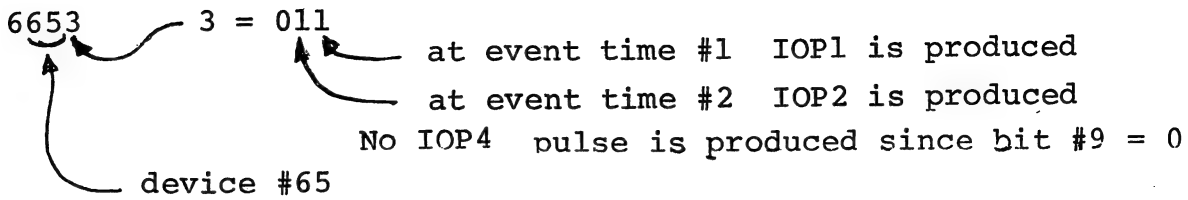
IOP4 6354

IOP1, IOP2 and IOP4 each have separately assigned connections at the computer's output to enable each pulse to be used for a unique operation at each peripheral device.

Write the instruction to command device #37 to operate at event time #2. Which IOP is accepted by the addressed peripheral device?

6372 IOP2

Write an instruction which in device #65, causes operation A to be commanded by IOP1 and operation B to be commanded by IOP2, 1.0μsec later.



Each of the IOP's (IOP1 and IOP2) will be routed via hardware in the peripheral device to command the required operations, A and B. For example IOP1 might clear a data register and IOP2 might start a measurement operation 1.0μsec later.

F-13

Most peripheral devices have a flag, usually comprising an RS Flip-Flop which is set to the 1 state when the peripheral device is ready to communicate to the computer and is in the 0 state when the peripheral device is busy or not ready. For example the teleprinter, device 04, has a flag which is cleared (Flag → 0) via the instruction TCF (6042). The same flag is set (Flag → 1) by the printer at the end of each 100 msec printing operation initiated by TPC (6044). In going to the 1 state the flag signals to the computer that another 8 bit ASCII code character can be accepted by the printer.
TCF (6042) and TPC (6044) may be microprogrammed to form TLS (6046) causing the sequence of (a) clear the flag

(b) Load the printer buffer and print the character

(c) Set the Flag

The computer tests the flag status via the instruction TSF (6041) which is interpreted as "Skip the next instruction if the Flag = 1". In the program below what is the second instruction executed if the flag = 0?

```
2200/6041
2201/5200
2202/6046
```

2201/5200 Since the flag = 0 no skipping occurs. Therefore the second instruction executed is 2201/5200 causing a jump back to the first line 2200/6041.

How does the computer get out of this loop?

The flag must go to 1, or in other words must be set, perhaps via the completion of some previous TPC (6044) operation.

When the flag goes to 1, what instruction is executed after 2200/6041?

2202/6046, causing the clearing of the flag and the printing of a character.

Details: When the flag = 1 the instruction 2201/5200 is skipped and the instruction 2202/6046 is executed. At the end of the printing operation the flag is automatically set. (Flag → 1). The computer only takes 4.25 μsec to execute the TLS (6046) instruction. The printing operation itself takes 100 msec. Hence the flag goes to 1 at the end of the 100 msec. period, perhaps while the computer is executing some other portion of the program, or perhaps while the computer is waiting in a loop for the flag to be set.

The following frames in section F are for those interested in the hardware aspects of ths skipping operation and are not mandatory for those interested only in programming.

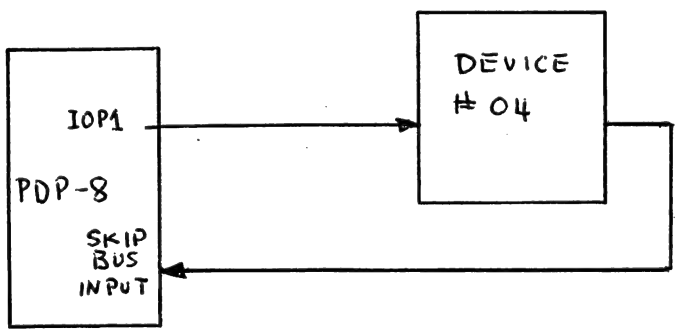
- The instruction 6041, which is the "Skip when flag = 1" instruction produces what IOP in what device #?

IOP1 in device #04, the teleprinter.

Will any other peripheral device accept IOP1 when 6041 is executed?

No, only device #04

So far IOP1 has been produced by the IOT instruction 6041 and accepted only by device #04. This IOP pulse is used to cause the computer to skip the next instruction in the program. This is effected by connecting this IOP1, after it is inverted in device #04, to the SKIP BUS on the computer. When the SKIP BUS which normally is in the 1 state is brought to 0, via in inverted IOP pulse, the program counter in the computer is incremented, causing the next instruction in the program to be skipped. Thus the inverted IOP1 pulse goes to 0 at the output of device #04 causing the SKIP BUS to be enabled.



Example

0410/6041	TSF	The computer is in a loop, 6041
0411/5210	JMP .- 1	5210, 6041, 5210 etc. until the
0412/6046	TLS	flag is ultimately set. Consequently the skip bus is enabled, the instruction 5210 is skipped over and 6046 (TLS) instruction is then executed.

The next question likely to be asked is how does device #04 detect its device code in the instruction 6041. The Memory Buffer contains the instruction being executed, eg. the instruction 6041. Memory Buffer bits #3 to 8 define the device code, and at the computer's output these 6 bits are available as logic levels both in true and complementary form (12 outputs total). To detect the device code in the instruction 6041 first note that 04 = 000100

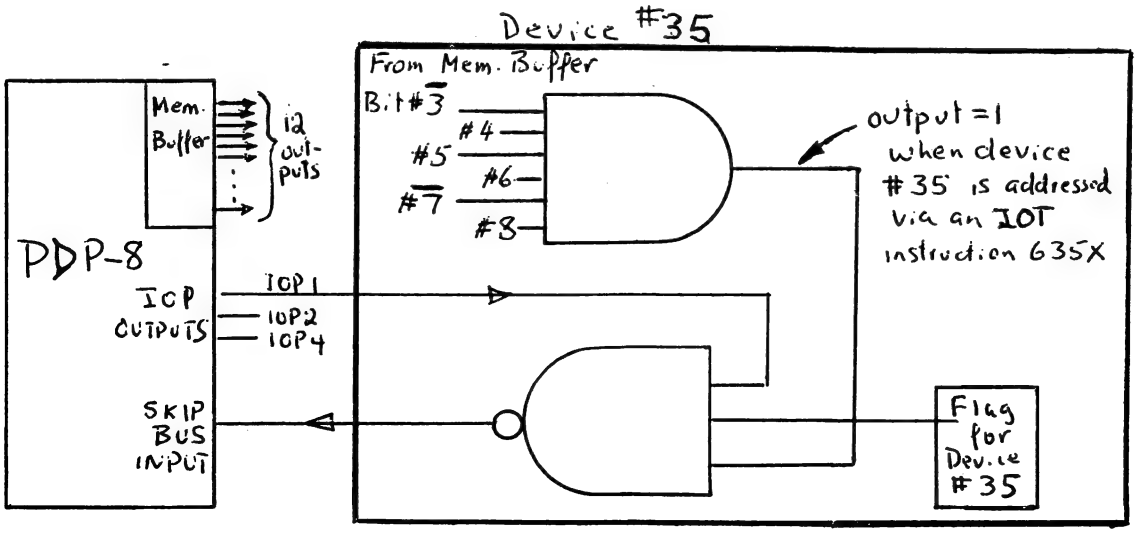
that is Bit #3, 4, 5, 7 and 8 are 0
Bit #6 = 1

A six input AND gate with the complement of bits #3, 4, 5, 7 and 8 and the true version of bit #6 as inputs will produce a 1 at its output if, and only if, 04 is the device code. Furthermore this 1 state will be unique since all other AND gates for other peripheral devices will be wired differently for their device code assigned.

Which of bits #3 to #8 are used in the true form and which are used in the complementary form for the AND gate in device #35.

First Note 35 = 011101
Bits 4,5,6 and 8 are used in the true form
Bits #3 and 7 are used in their complementary form since their state is 0 when device code 35 is called in an instruction such as 6352.

35 = 011 101
Bit #3 — Bit #8
Bit #7



PROGRAM INTERRUPT

OBJECTIVES

In this section you will write instructions which avoid the long delays encountered in waiting for flags in peripheral instruments to be set.

An input/output program can be very inefficient if the computer spends most of its time waiting for the flag on a peripheral instrument to be raised, before the required data can be transferred between the computer and the peripheral instrument.

To circumvent this problem the "program interrupt" mode of operation is often used. "Program interrupt" permits a peripheral device to signal to the computer, via its flag, that it is ready for service, that is, it is ready to transfer data to or from the computer. This is accomplished by permitting the peripheral's flag to enable the INTERRUPT BUS, a computer input line. Enabling of input bus produces two automatic actions.

(1) The Interrupt facility is disabled (equivalent to IOF).

(2) The next instruction address of the main program is stored by the computer in address 0000. In the next address of the subroutine (0001) a jump instruction is often stored by the programmer to direct the computer to the appropriate memory address where the service routine resides. The last instruction of the service routine will be 5400 (JMP I 0000) returning the computer to the main program which was interrupted by the peripheral instrument's flag. This sequence of events is very similar to the execution of a subroutine.

Open up Panel G at the back of these notes. Here a portion of the main program is shown. Note that the ION instruction (6001) arms the interrupt line to permit the computer to accept an interrupt at any time during the main program. During the execution of 3025 assume that the interrupt flag on the teletype is raised due to the teletype's requirement for a new 8 bit ASCII word from the computer.

Before responding to the interrupt the computer completes the instruction being executed, 3025 in this case. Then an IOF followed by a JMS 0000 (4000) instruction is executed automatically.

What address is deposited in 0000?

0622, the next address in the main program.

After the automatic execution of 4000 by the computer, what does the next instruction cause?

The instruction in 0001 causes an indirect jump via the pointer address 0023 to the effective address 3260.

G-3

The next instruction, 1271, provides data for the teleprinter. The following instruction, 6046, causes the start of the 100 msec printing operation. Why does no 6041 (TSF) and 5262 (JMP.-1) flag checking operation precede the 6046 instruction?

These two instructions are not needed because the flag is already set, causing the service routine to be executed via the program interrupt. Since the teleprinter is ready for the data, the 6046 instruction may be executed immediately. This means that data may be provided for the teleprinter in much less than 25 microseconds rather than having the computer wait for up to 100 msec when the flag status check rather than the interrupt mode of programming is used.

G-4

The instruction ION (6001) preceding the last instruction of the service routine arms the interrupt circuitry permitting another interrupt to be accepted. This arming action is internally delayed until after the next instruction is executed. This permits control to return to the main program before another interrupt can be accepted. How does the instruction 5400 cause a return to the main program? Be specific.

5400 is an indirect jump via the pointer address, 0000, where the effective address for the jump to the main program is stored. Execution of 5400 returns control to the main program, to the next instruction to be executed when the interrupt occurred, that is to 0622/1047.

Sometimes the AC and link data must be preserved before the service operations in the subroutine are executed. Rewrite the first 3 lines of the service routine, causing this storage operation. If you have problems with the link storing operation, a hint is given in the next frame.

3260/3274 - (DCA AC) - typical but not a unique answer
storing the AC data in address 3274

Hint: For the link storage operation first use a "Rotate" instruction to move the link data into the AC.

3261/7004 - (RAL) Rotate the link left. This places the link data in bit #11 of the AC. (RAR also is acceptable, with the link data going to bit #0)

3262/3275 - DCA LINK Again a typical answer but not unique, storing the data in 3275.

The rest of the instructions for the service routine will have to move two locations down the list, eg. 3263/1271.

A reverse operation will be required at the end of the service routine to restore the link and then the AC. Write the instructions assuming that the AC and link storage operations are as shown in the above frames. Let this part of the program start at address 3320.

3320/7200 - (CLA)
3321/1275 - (TAD LINK)
3322/7010 - (RAR) (Use RAL if RAR was used in the previous frame)
3323/1274 - (TAD AC)

When 2 or more devices are permitted to cause an interrupt an identification operation is necessary after the interrupt occurs to determine which peripheral device requires service. In Panel G-2 a "skip-chain" is used to test the flag of each peripheral device until a raised flag is found, at which point the appropriate service routine is entered. Assume that device #35 has an IOT instruction 6351 which is interpreted as "Skip if device #35's flag = 1".

What sequence of instructions will be executed beginning at address 3200 in Panel G-2 if device #35 caused the interrupt.

6041 - Test flag 04, which is <u>not</u> up at this time.
7410 - Skip the next instruction (5250)
6351 - Test flag 35, which is up at this time.
5300 - Jump to service routine for 35.

The service routine for device 35 will end with ION (6001) and then 5400 (JMP I 0)

Congratulations! You are finished. For more details on programming techniques refer to "Introduction to Programming" published by Digital Equipment Corp.

PDP-8 INSTRUCTIONS

000 XXX XXX XXX = AND = perform the logical AND, bit by bit, using the AC and the contents of memory address XXX XXX XXX.
Leave the result in the AC.

001 XXX XXX XXX = TAD = Add to the AC the octal data in address XXX XXX XXX.
Twos' complement arithmetic will be used for negative numbers.

010 XXX XXX XXX = ISZ = Increment (Add 001 to) the contents of memory address XXX XXX XXX and if the contents then is zero, skip the next instruction in the program.

011 XXX XXX XXX = DCA = Deposit the contents of the AC in address XXX XXX XXX and then clear the AC. The deposit action overwrites data previously in the memory address.

100 XXX XXX XXX = JMS - Jump to the subroutine located at address XXX XXX XXX and store the next address of the main program in the first address of the subroutine.

101 XXX XXX XXX = JMP = Jump to memory address XXX XXX XXX and execute the instruction stored there.

110 XXX XXX XXX = Input/Output Transfer

111 XXX XXX XXX = Operate Microinstructions. For details see Panel B-2.

Panel B-2

PANEL B-2

GROUP 1 OPERATE MICROINSTRUCTIONS

NOP	7000	no operation
CLA	7200	clear AC
CLL	7100	clear link
CMA	7040	complement AC (one's complement)
CML	7020	complement link
RAR	7010	rotate AC and link right one
RAL	7004	rotate AC and link left one
RTR	7012	rotate AC and link right two
RTL	7006	rotate AC and link left two
IAC	7001	increment AC

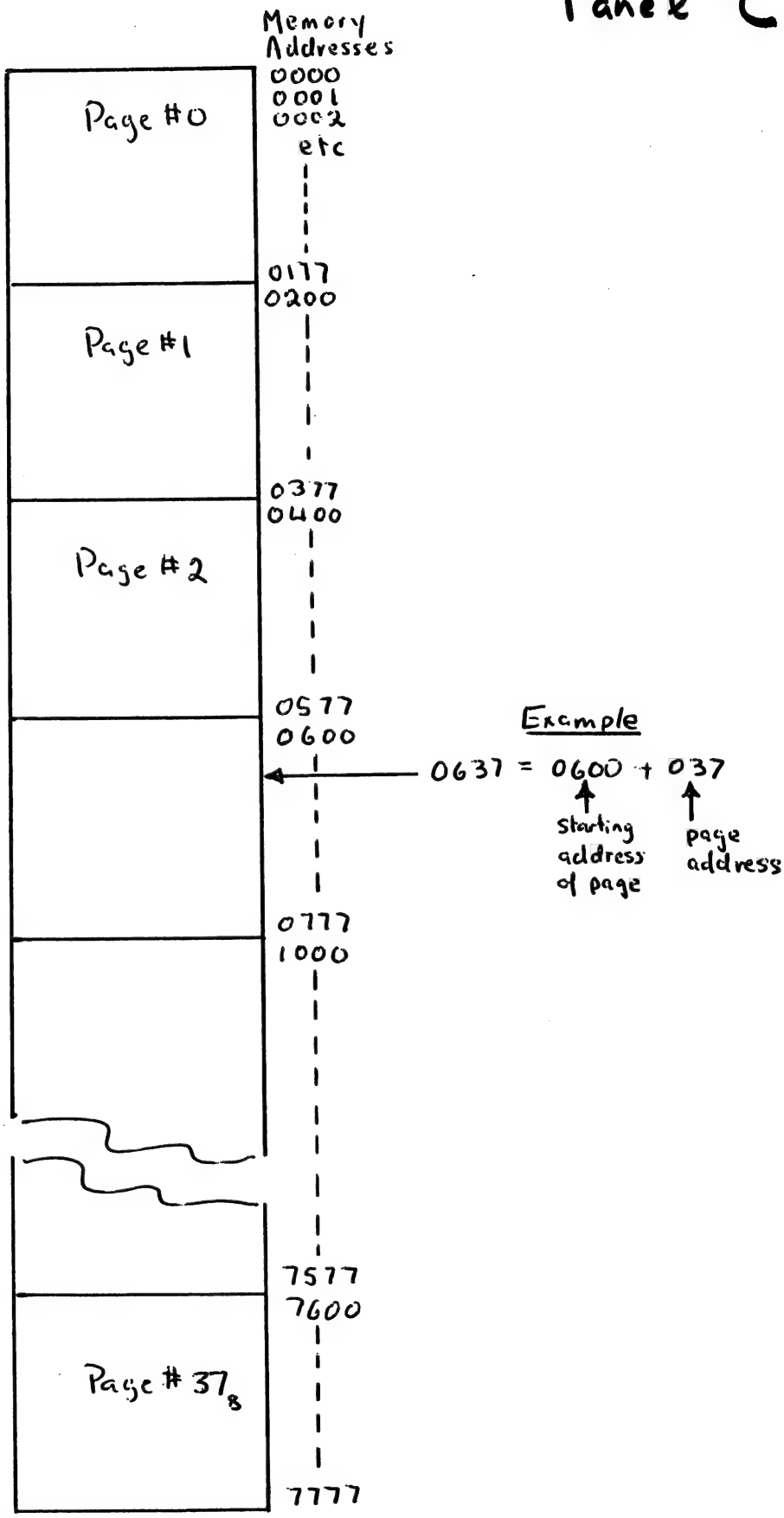
GROUP 2 OPERATE MICROINSTRUCTIONS

SMA	7500	skip on minus AC
SZA	7440	skip on zero AC
SPA	7510	skip on plus AC
SNA	7450	skip on non-zero AC
SNL	7410	skip on non-zero link
SZL	7430	skip on zero link
SKP	7410	skip unconditionally
OSR	7404	inclusive OR, switch register with AC
HLT	7402	halts the program
CLA	7600	clear AC

COMBINED OPERATE MICROINSTRUCTIONS

CIA	7041	complement and increment AC (two's complement)
LAS	7604	load AC with switch register
STL	7120	set link (to 1)
GLK	7204	get link (put link in AC bit 11)
CLA CLL	7300	clear AC and link
CLA IAC	7201	set AC = 1
CLA CMA	7240	set AC = -1
CLL RAR	7110	shift positive number one right
CLL RAL	7104	shift positive number one left
CLL RTL	7106	clear link, rotate 2 left
CLL RTR	7112	clear link, rotate 2 right
SZA CLA	7640	skip if AC = 0, then clear AC
SZA SNL	7460	skip if AC = 0 or link is 1, or both
SNA CLA	7650	skip if AC $\neq$ 0, then clear AC
SMA CLA	7700	skip if AC < 0, then clear AC
SMA SZA	7540	skip if AC $\leq$ 0
SMA SNL	7520	skip if AC < 0 or link is 1, or both
SPA SNA	7550	skip if AC > 0
SPA SZL	7530	skip if AC $\geq$ 0, and if the link is 0
SPA CLA	7710	skip if AC $\geq$ 0, then clear AC
SNA SZL	7470	skip if AC $\neq$ 0 and link = 0

Panel C


Memory Addresses
0000
0001
0002
etc
Page #0
0177
0200
Page #1
0377
0400
Page #2
0577
0600
0777
1000
7577
7600
Page #37₈
7777
Example
0637 = 0600 + 037
Starting address of page
page address

Panel D

2621/7200
2622/1471
2623/7402
0071/3643
3643/2653

1460/7200
1461/1567
1462/3453
1463/7402
0167/6653
0053/2433
6653/1205

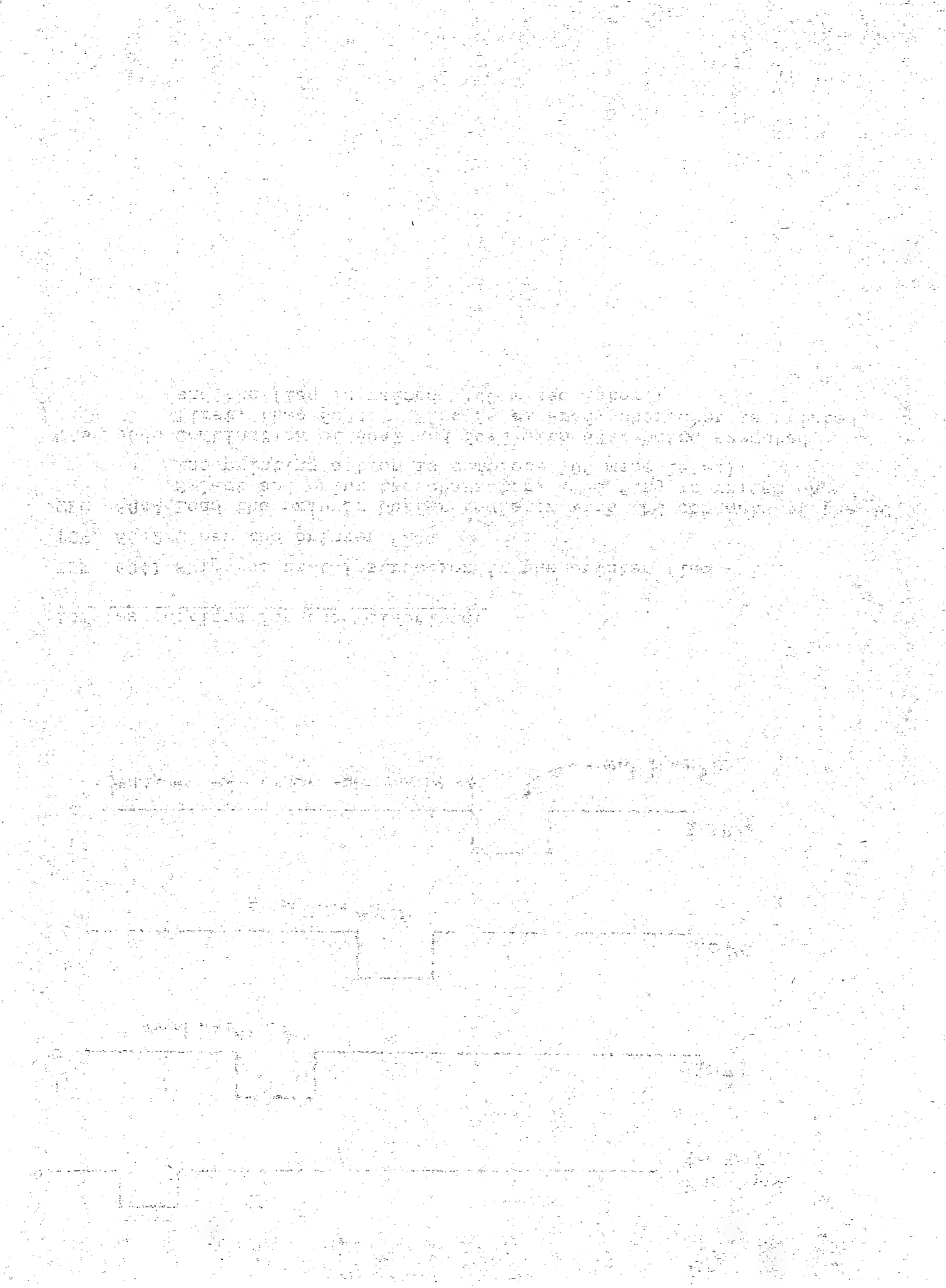
Panel E

```
     *1400
     CLA        - 1400/7200
     TAD A      - 1401/1271
     JMS B      - 1402/4060
     TAD C      - 1403/1272
     HLT          1404/7402

A,   0005         1471/0005
C,   0003         1472/0003

B,   0000         0060/0000
     CIA          0061/7041
     JMP I B      0062/5460
```

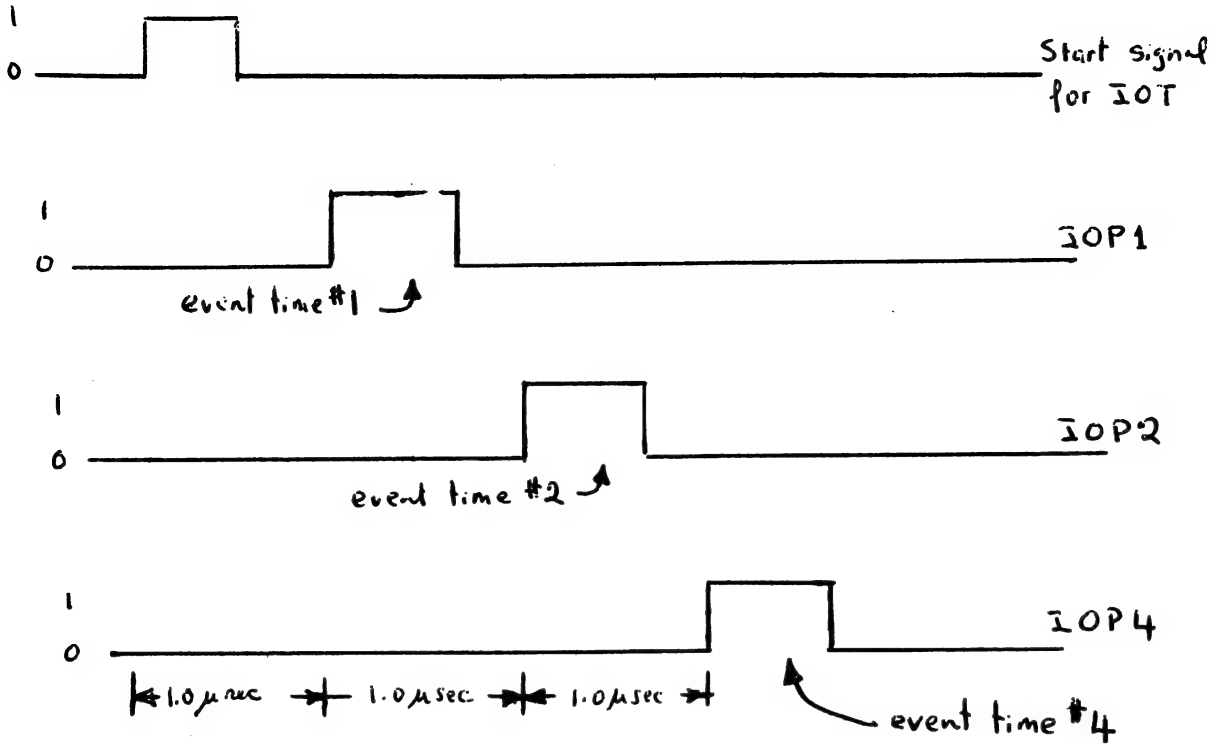
Main Program

Data for main program

Subroutine



Panel F



IOT Instructions for the Teleprinter

TSF 6041 Skip the next instruction if the printer flag = 1

TCF 6042 Clear the printer flag.

TPC 6044 Load the printer buffer register with the contents of the AC Select and print the character. (The flag is raised when the printing action is complete 100 msec later).

TLS 6046 Combination of 6042 and 6044 with 6042 being executed first, then 6044. (Flag is cleared, character is printed and the flag is raised 100 m sec later.)

Panel G

Panel G-1

0600/6001 INTERRUPT ON This arms the interrupt circuitry to permit an interrupt to be accepted.

0620/1043 Add contents of 43
0621/3025 Store in 25 ←
0622/1047 Add contents of 47 to the AC

ASSUME THAT AN INTERRUPT OCCURS AT THIS TIME DUE TO THE PERIPHERAL'S FLAG BEING SET. TIMING OF THE FLAG SETTING IS INDEPENDENT OF THE MAIN PROGRAM.

0000/0000
0001/5423
0023/3260
3260/7200
3261/1271
3262/6046
3263/6001
3264/5400
3275/0266

Panel G-2

0023/3200

3200/6041 - Skip if flag 04 = 1
3201/7410 - SKP Skip the next instruction
3202/5250 - JMP - Service routine for teleprinter
3203/6351 - is flag 35 up, skip if it is
3204/5207 - Jump to 3206
3205/5300 - Service routine for device 35
3206/6001 - ION
3207/5400 - JMP I 0 { False alarm Return to the main program.